A Large-Scale Distributed Federated Experimental Facility Offering Simulation Services for Next Generation Wireless Networks

Georgios Kormentzas[*]

Department of Information and Communication System Engineering, University of the Aegean
Samos, Greece

[*]gkorm@aegean.gr

Abstract

Efficient coordination and management of the radio resources in heterogeneous wireless networks is a key requirement for the realization of the next generation 5G wireless networks. In this context, experimental infrastructures that allow for performance evaluation of inter-system optimization algorithms are of great value. The paper presents a distributed software platform that enables inter-system operations and interworking among geographically distributed clusters simulating different radio access technologies. The functionality of the key building blocks of this distributed virtual testbed is discussed together with the respective design and implementation details.

Keywords

Distributed Simulation; Event-based Middleware; SOAP; Wireless Network

Introduction

The convergence of circuit-switched and packet-switched networks and, at the same time, the increase of available data rates of wireless links have led to the rapid evolution of wireless networks that integrate heterogeneous Radio Access Technologies (RATs) in composite network topologies. In such environments, terminals are equipped with multiple radio interfaces and are capable of attaching to different wireless access networks either alternatively or concurrently. Efficient Radio Resource Management (RRM) schemes that allocate the radio resources among the different networks in an optimized manner would be definitely an asset towards the farther evolution of the wireless networks.

The various inter-system optimization schemes proposed in the international literature before their deployment at real wireless settings have to be validated usually through a set of simulation scenarios. Mature standalone network simulators provide rich APIs and tools and have been extensively used by researchers. However, their standalone approach does not allow for flexibility, language independence and easy implementation of a new algorithm. On the other hand, till now, distributed network simulation is mainly considered as provision of distributed simulation frameworks [Grau 2008, Hu 2005, Riley 2004], rather than as provision of complete simulation engines for heterogeneous network technologies, where the RRM algorithms refer to.

Considering the above, two main problems arise in the area of simulation of next generation wireless networks: (a) The rapid evolution and the heterogeneity of different wireless technologies makes difficult the development of integrated simulation engines, capable of performing detailed simulation of every wireless access technology in all layers of the OSI model. (b) Implementation of an algorithm in a simulation testbed requires from the owner to have deep knowledge of the simulator and the language that was implemented.

Motivated by the need for a testbed that allows for simulations of different radio access technologies and, at the same time, provides facilities for testing optimization algorithms without the need of modifying the simulation engine, this paper proposes a Virtual Distributed Testbed (VDT) that interconnects standalone, physically and language independent software and hardware simulators, each one simulating a different radio access technology. The whole approach is motivated by the following two key ideas: (a) Instead of developing a composite simulator from the scratch, a middleware and a control unit are provided for distributed interconnection of already available and future developed simulators, allowing full exploitation of the capabilities of each one, and (b) Instead of considering an RRM algorithm as a part of the simulator, it is considered as a separate entity, totally independent of the whole simulation testbed, allowing for easy implementation of RRM algorithms.

The advantages of the above approach compared to other distributed network simulation testbeds are: (a) Distributed sharing of processing recourses, (b) Scalability by means of easily attaching new hardware and software testbeds simulating forthcoming technologies, (c) Easy implementation of RRM algorithms without knowledge of the simulation testbed, as the whole simulation engine is transparent to the independent entity that implements the algorithm, and (d) Geographical independence of simulators, allowing attaching hardware testbeds that cannot be moved.

The rest of the paper is organized as follows: Section II describes the architecture of the distributed simulation testbed. Section III deals with general design and management issues. In Section IV design and implementation details are given for each functional entity separately. Finally, Section V concludes the paper.

Architecture of the Virtual Distributed Testbed

The aim of the proposed testbed is to integrate independent standalone geographically distributed software simulators and hardware testbeds so as to allow for inter-system optimization studies in heterogeneous networking environments. Towards this direction, two main functional requirements need to be addressed. The first is the design of a distributed approach for simulation of composite wireless networks by interconnecting existing simulators of diverse radio technologies running on different platforms. The second has to do with the incorporation of facilities for joint RRM functions.

These requirements can be realized by the functional architecture depicted in Fig. 1. This architecture relies on the identification of four conceptual levels for the distributed virtual testbed and the definition of discrete building blocks residing at these levels. At the topmost level, the end-user interface provides access to the VDT Controller, which is responsible for connecting the VDT platform with the end user interface, called VDT Editor. Using this interface and the facilities provided by the VDT Controller (e.g., identification of available simulators, retrieval of simulator parameters etc.) end-users create simulation plans and submit them for execution (plans are created through a Scenario Editor, which is part of the VDT Editor). Additional functionalities of the VDT Controller include management of user authentications and permissions, retrieval of past simulation plans from the repository, validation of simulation plans, scheduling of the simulation instance execution and storage of simulation plans to a repository for future use.

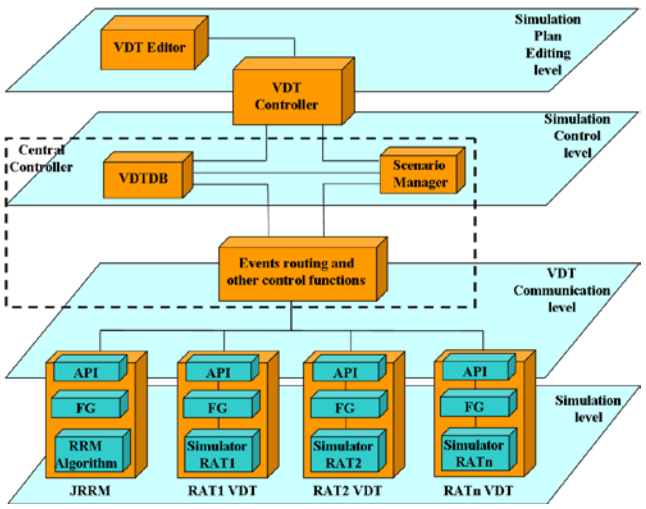


FIG. 1 THE PROPOSED VDT ARCHITECTURE

When a simulation is ready for execution, VDT Controller hands over control to the Central Controller. The latter is composed of the VDT Database (VDTDB) module responsible for storing simulation results, simulation plans and simulator-related information, and the Scenario Manager that controls the whole simulation, undertaking actions like terminal management, service and traffic stream management and simulator clusters time management and synchronization.

At the lowest level of the VDT architecture, there are a number of modules responsible for simulating radio technologies and the module that implements the joint RRM algorithm (JRRM). Legacy simulators are attached to the distributed virtual platform by forming RAT VDT modules. These modules are comprised of two entities: a) the actual simulator (in general the Federated Service) and b) the Federated Gateway which manages the simulation cluster and translates simulator-specific messages to VDT messages and vice versa. The same structure stands for JRRM as well, where, instead of the simulator, the actual RRM algorithm exists along with the corresponding Federated Gateway. However, the proposed testbed provides some RRM algorithms already implemented in the Central Controller. For the case of these algorithms, the JRRM module is considered as part of the Central Controller. The notion of the Federated Gateway and, generally, the approach of federation were adopted from the HLA standard [IEEE Std]. However, HLA is a general purpose architecture, where part of its functionality does not always fit to the simulation engine's needs. Hence, instead of implementing the communication layer in compliance with HLA, a different design approach was adopted in order to fit in the needs of a simulator for heterogeneous wireless technologies.

In the context of VDT, the communication is achieved through the definition of appropriate events. These events are implemented using SOAP over HTTP. The control of the events and their routing towards the distributed components is undertaken by the Central Controller. For each potential RAT and JRRM VDT module, the communication with the Central Controller is achieved through the VDT Module API, which provides the interface (functions and parameter definitions) through which VDT modules communicate with the components of the Central Controller.

The definition of the SOAP based events and the provision of the SOAP-based middleware and API are the key functions that allow for easy attaching of new simulators and algorithms. Indeed, the owner of the simulator or the algorithm needs only to know the appropriate events to communicate with the Central controller and the parameters embodied in each one. From the implementation point of view, the owner needs only to use the provided API and create the Federated Gateway that calls the desired function on each event received.

General Design of the Virtual Distributed Testbed

Based on the architecture described above, this section presents in detail the design of the proposed virtual distributed testbed. We start by addressing crucial management issues like time management and service management. Next, based on the architecture presented in Section II, we specify in detail all the necessary events for communication between the distributed modules.

Addressing of Management Issues

In the following, we concentrate on management issues in order to efficiently control the execution of the distributed simulation. Although the discussion is focused on inter-system optimization functions like initial RAT selection and vertical handovers, other joint RRM functions (like admission control and load balancing) can be supported as well.

1) Time Management

Time management in the proposed testbed is achieved by a simple yet efficient scheme that guarantees that all connected simulators are synchronized at specified time instances (i.e., they have simulated up to a given time). More specifically, the simulation time is divided in a number of equal time steps, the duration of which is defined by the user. The overall synchronization of the simulation is performed by the Scenario Manager by a process where, for every time step, it indicates to the simulators to simulate it and waits until all the simulators have simulated the specific step. Every simulator is responsible for generating simulation results for the interval simulated and for storing them to the VDTDB. When all simulators have finished simulating the specific time step, the Scenario Manager can proceed to RRM actions, session initiation/termination actions, or indicate to the simulators to simulate the next step. Note that, during the simulation of a time step, no other action (e.g. handover, start of new session, association of a mobile) is taking place. All these actions take place between simulations of time steps.

However, considering the above, the time step duration must be selected in a way the simulation is stopped at time instances where optimization actions or session initiations and terminations take place. This means that the simulators need to be synchronized both when sessions are setup/end and when RRM algorithms are executed. The requirement to address synchronization at session setup/termination points results in an "irregular" set of synchronization points (in the sense that session durations are not always constant). RRM algorithms, on the other hand, can be reasonably assumed to be executed on a regular basis (e.g. every few seconds) or during longer time periods. This tradeoff can be balanced by setting the start of sessions, the session ON and OFF duration and the RRM execution time instances as multiples of the time step duration.

2) Terminal Management

Considering terminal management, the VDT platform addresses two types of terminals: a) mono-mode terminals which are connected to and managed by only one simulator and b) multi-mode terminals that have connectivity to several simulators (each one simulating the radio technology of a given interface). At the system level (composite simulator level) every mobile terminal is allocated a unique global identifier. For multimodal terminals, however, there is an additional need to identify the mobile in all systems it is supposed to have access to. This introduces a second level of mobile indexation inside RAT VDT modules. The mapping between the global mobile identifier and the simulator-specific identifier is performed by the Scenario Manager through the structure illustrated in Fig. 2a.

The presented mapping is based on the concept that (virtual) multi-mode terminals are identified by many instances, each one of which applies to a different simulator. The global IDs are allocated by the Scenario Manager in a centralized manner while the local ones are assigned by the corresponding simulators upon terminal registration in the simulated radio access system. A local ID of zero means that the mobile is not associated with the specific RAT yet. When a mobile is added to a RAT VDT module, the module assigns a local ID and informs the Scenario Manager of the ID assigned. After that, the Scenario Manager refers to the RAT for the specific mobile through its local ID.

3) *Service Management*

The VDT testbed supports several types of services, all of which comply with a common service description. Service description in the proposed testbed is defined through the term 'service'. Examples of services include Voice over IP, HTTP, Near Real Time Video, FTP as well as more abstract services corresponding to constant and variable bit rate traffic models. Each one of the specified services is identified by a unique ID and a set of traffic parameters and a set of QoS requirements (e.g. minimum rate, tolerable packet loss and delay etc). The traffic parameters and the QoS requirements of each service can be defined by the user. The mapping of each service ID to its corresponding traffic and QoS parameters is performed in the Scenario Manager through the mapping table depicted in Fig. 2c. This mapping table also exists in each RAT VDT module. Default parameters exist for all supported services, so when the Scenario Manager wants to indicate to a RAT module to start a service, it refers to the latter through the service ID. However, service parameters can change, as set by the user. In this case, the Scenario Manager must refer to the service through its ID and the name/value pair of any parameter that is different from its default value.

(a)

Global ID	RAT1 Local ID	RAT2 Local ID	...	RATN Local ID
1	2	5	...	0
2	0	3	...	3
3	1	0	...	6

(b)

Session ID	Service ID	Stream Parameters					Mobile Params		RAT Params	
		start	ON	OFF	On Dstr	Off Dstr	Src Mobile ID	Dst Mobile ID	SrcRat	DstRat
1	2	5	3	6	Const	Exp	2	0	1	0
2	3	10.1	6	3	Exp	Exp	3	6	1	2
3	2	4	2.4	7.5	Exp	Exp	0	4	0	4

(c)

Service ID	Service descr.	Service traffic Parameters					QoS Parameters	
		pkt size	Pkt dstr	file size	File size dstr	...	delay	...
1	CBR	1024	Const	-	-	...	10	...
2	VBR	512	Exp	-	-	...	10	...
3	FTP	1024	Logn	4000000	Exp	...	200	...

(d)

Mobile ID	Results			
	Throughput	utilization	Delay	...
1	64	0.064	73	...
2	128	0.128	11	...
3	64	0.064	47	...
0	256	0.256	-	...

(e)

Time	Mobile 1		Mobile N		RAT 1		RAT N	
	Throughput	...	Throughput	...	Throughput	...	Throughput	...
1	64	...	128	...	32	...	512	...
2	72	...	101	...	35	...	768	...

FIG. 2 MANAGEMENT TABLES

Services are executed to the mobiles as applications. Each mobile can execute one or more services simultaneously in different RATs. Service execution in the proposed testbed is referred through the term 'session'. More specifically, a session is an execution of a service for a specific time interval between specific mobiles. For example, a video streaming session with duration of 10min and idle time of 20min means that every twenty minutes a user opens its browser and watches a video that lasts 10 minutes. The parameters of the video streaming service are defined in the service mapping table for the ID of the specific service.

Sessions can be initiated between mobile nodes in different RATs or between mobile and fixed nodes (i.e. nodes that are supposed to belong to a wireline infrastructure outside the wireless simulators). Every session is bound a specific service ID, which is the ID of the service to be executed. Moreover, every session is bound to source and destination mobiles. If the traffic of the session is bidirectional however, this is mapped to two different sessions.

Session arrivals are repeated in terms of an ON/OFF model defined through the parameters of session start time, session duration and intersession idle time. The last two parameters can be constant or exponentially distributed. Session start time is the simulation time instance where the execution of the service (application) starts. Session duration is the duration of the service execution (ON period of execution). After the end of the execution, the terminal remains idle for an interval specified by the intersession idle time (OFF period of execution). After the end of the idle interval, the service starts executed again.

When a new session is setup, it is assigned a unique ID. Several sessions (applications) can run on a mobile using any of the available radio access technologies. The mapping of each session to a service, source/destination mobiles and source/ destination RATs is performed in the Scenario Editor and in the Scenario Manager, through the mapping table depicted in Fig. 2b. In the Scenario Editor, a similar table exists where the user sets the scenario story through definition of sessions. From the scenario Manager point of view, this table is needed because session mapping to source/ destination RATs is dynamic, as, when RRM actions take place, execution of a service can be transferred to another RAT.

4) *Simulation Results Management*

After the simulation of a time step, each RAT VDT module must send the results concerning this time step to the VDTDB. The VDTDB must store these results as records to the database. The results are sent to the VDTDB through a message containing a results table similar to the one depicted in Fig. 2d. Note that the mobile IDs refer to local mobile IDs.

The results provided by the simulators can be requested from the JRRM VDT module in order RRM decisions to be taken. The VDTDB provides the results in a sub-table of the one described above. The JRRM can request information about the whole system, so only the last record is provided, or about a specific mobile, so a specific record is provided only. It is responsibility of the VDTDB to select and join the appropriate records from appropriate RATs depending on which RAT the mobile is attached at a specific time instance.

Moreover, the VDTDB must keep tracking of the results for a specific mobile in all RATs it is attached during the simulation time. In order to achieve this, a join table exists in the database, similar to one depicted in Fig. 2e, constituted of separate sub-tables the fields of which are time, mobile or RAT ID and results (throughput, delay e.t.c.).

Specification of Events

Based on the architecture described in Section II and the management functions described in the previous subsection, in the following we describe the events that are required to realize simulations of composite wireless networks and test initial RAT selection and inter-system handover algorithms on the proposed VDT platform. These events, which are grouped according to their functionality, are exchanged among the Scenario Manager and the VDTDB modules of the Central Controller and the geographically distributed RAT VDT and JRRM modules. In the application level, this event is a function that is transformed to a SOAP message in the VDT communication level (see Fig. 1).

1) *Initial Configuration and Time Synchronization*

Each RAT VDT module has specific configuration parameters that must be initialized before the simulation starts. These parameters are provided by the Scenario Manager through the ProcccessConfiguration event. (This is the only event the format of which is simulator-dependent.) After the initial configuration of the simulators, the Scenario Manager sends a Start event to all RAT VDT modules to start the simulation. Time synchronization in the VDT platform is achieved through the ProcccessTime and ProcccessTimeReply events. The former is sent by the Scenario Manager to the simulators and allows them to simulate for the specified time interval (i.e., simulation step delineated by the start time and stop time float parameters). The stop time can coincide with the next time in future that the optimization algorithms will be executed. The ProcccessTimeReply event is sent as a reply to the Scenario Manager, as soon as the specified interval has been simulated and the simulation results concerning this interval have been sent to the VDTDB. After the simulation is complete, a Stop event is sent to all VDT modules, indicating that the simulation is completed.

2) *Terminal and Service Configuration*

A number of events are defined to let simulators perform specific actions for terminal and service management. When a new mobile must be added to a simulator, an AddMobile event is sent by the Scenario Manager. When the mobile is added, an AddMobileReply is sent back to the Scenario

Manager containing the local ID of the new added mobile (see next Section about global and local mobile IDs). Since the the AddMobileReply event is sent, the newly added mobile will appear in the simulator as an idle node; it will not yet produce or receive any traffic.

Configuration of the profile of the traffic that will be transmitted or received by a mobile within a RAT is achieved through the SendStreamToMobile event that is sent by the Scenario Manager to the RAT VDT module. This event contains an integer parameter that defines the ID of the service to be executed by the mobile. After the configuration of the traffic is completed, a SendStreamToMobileReply is sent back to the Scenario Manager indicating the successful configuration of the traffic profile of the specific mobile. Still, the mobile node remains idle, until an event is received to activate the execution of the service configured.

Traffic activation and deactivation is achieved through the StartTxToMobile and StopTxToMobile events. Note that the StartTxToMobile event is not combined with the SendStreamToMobile event in order to avoid multiple configurations of the same service in cases of frequent handovers). The StartTxToMobile is sent by the Scenario Manager to the RAT VDT module, incorporating an integer parameter with the local ID of the mobile that should start transmitting traffic according to the profiles that were previously configured. The StopTxToMobile event, on the other hand, indicates that a specific mobile should stop transmitting traffic. These events are complemented by appropriate replies (StartTxToMobileReply and StopTxToMobileReply), indicating that the activation/ deactivation of service execution is completed successfully. Note that when a StopTxToMobile event is received from a RAT VDT module, the corresponding node stops executing any service but remains connected to the RAT, receiving any control traffic like beacon messages.

Finally, when a mobile is to be completely disconnected and disassociated from a RAT, a RemoveMobile event is sent by the Scenario Manager to the RAT VDT module, incorporating an integer parameter with the local ID of the mobile to be removed. In this case, the mobile is completely removed from the simulator (all records for this terminal are deleted and all resources are free). A RemoveMobileReply event is sent back to the Scenario Manager as a reply, indicating that the mobile is removed successfully.

3) Manipulation of Simulators' Results

Each simulator is responsible for producing simulation results for the time interval that was asked to simulate. After the end of the simulation interval and before sending the proccessTimeReply to the Scenario Manager, each RAT VDT module sends an UpdateValues event to the VDTDB containing all the simulation results for the specific time interval. The results are embodied in the message as a single parameter (a table, the format of which is described in Section III) and stored in the database after the reception of the event from the VDTDB. Afterwards, the VDTDB sends an UpdateValuesReply event, indicating the reception of the results and their storage in the database. After the reception of the UpdateValuesReply event, the RAT VDT module sends the ProccessTimeReply event to the Scenario Manager.

The results stored in the VDTDB through the UpdateValues event are of interest to both the Statistics and the JRRM modules. In order to achieve proper communication between the JRRM and the VDTDB modules two events have been specified. The GetSystemInformation event is sent from the JRRM to the VDTDB, requesting system parameters (e.g. nominal data rate), and "system level" simulation results (e.g. total load, number of connected terminals) for a specific RAT, while the GetMobileInformation event requests detailed information (e.g. throughput, packet error rate, signal strength etc) for a specific mobile. The results, which also provide facilities to identify terminal and session mapping, are provided through the GetSystemInformationReply and the GetMobileInformationReply events respectively.

4) Control of JRRM VDT Modules

JRRM VDT modules can implement one of the following RRM functionalities; admission control for initial RAT selection and resource optimization through vertical handovers.

In the case of admission control, when a mobile asks for execution of a new service, an AdmissionRequest event is sent by the Scenario Manager to the JRRM module asking for the latter to decide for the acceptance of rejection of the service in the composite network. Apart from the service's ID and QoS requirements, the message also includes a list of simulated networks that are candidates for accepting the new service. Upon the reception of the AdmissionRequest event, An AdmissionReply event is sent to the Scenario Manager from the JRRM module, indicating the acceptance or rejection of the service. In the former case, the ID of the RAT in which the service will start is indicated.

In the case of resource optimization, this takes place in regular time intervals specified by the user. Every such interval, the Scenario Manager sends a PostProccessHandover event to the JRRM module, in order for the latter to report whether actions must be taken for efficiently reallocating sessions between the networks. The output of the resource optimization algorithm is sent to the Scenario Manager through the PostProccessHandoverReply event, which embodies a table with the sessions that will be subject to handovers and the target RAT VDT module for each one of them. The session reallocations themselves are performed by the Scenario Manager.

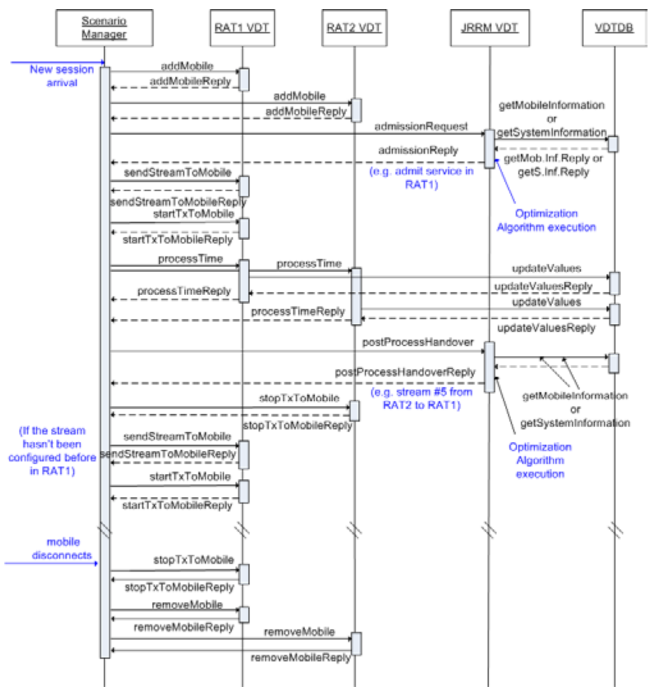


FIG. 3 SEQUENCE OF EVENT FOR CALL ADMISSION CONTROL AND RESOURCE OPTIMIZATION

5) *Events Sequence Example*

The functionality of the events presented above will be made clearer through the brief analysis of the steps that are required to realize initial network selection and resource optimization. In the event sequence diagram presented in Fig. 3 it is assumed that a composite network consists of two radio segments (simulated through VDT RAT1 and RAT2 modules) and that a normal ProccessTime/ ProccessTimeReply round has been performed just prior to the arrival of a new session.

When a service is initiated, a new mobile is added to each simulator through the corresponding AddMobile and SendStreamToMobile events. Assuming that the new service can be supported in both RATs, an AdmissionRequest event is sent to the initial network selection JRRM module. The latter retrieves information from the VDTDB in order to take decision, through the GetSystemInformation or getMobileInformation events. The decision is then propagated to the Scenario Manager through the corresponding reply event, which initiates service activation to the new node by sending a StartTxToMobile event. Upon the traffic configuration is over, the Scenario Manager sends ProcessTime events to the simulators in order for them to simulate for a specific time interval. After the time interval is simulated, each simulator reports the results to the VDTDB by sending an UpdateValues event. Upon the reception of the UpdateValuesReply event from the VDTDB, each simulator sends a ProcessTimeReply to the Scenario Manager in order to indicate that the time interval was simulated successfully.

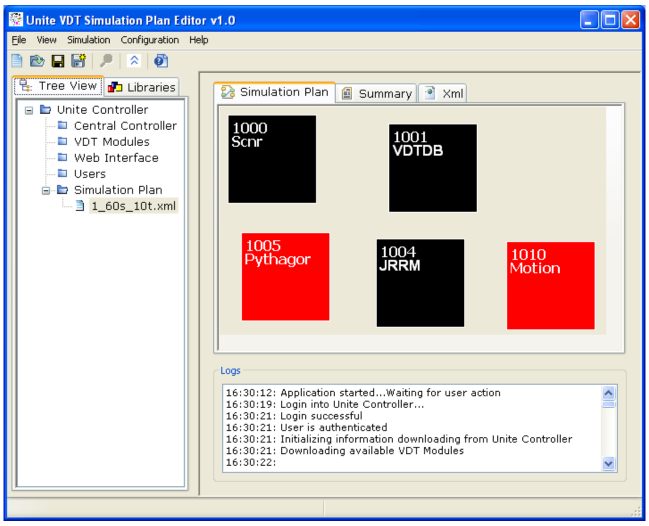
Afterwards, the resource optimization algorithm is executed. The execution is initiated by the Scenario Manager, which sends a PostProcessHandover event to the JRRM module. The decision that is propagated back to the Scenario Manager (through the PostProcessHandoverReply event) is implemented by appropriate StopTxToMobile, SendStreamToMobile and StartTxToMobile events sent to the past and future VDT RATs. Finally, if a mobile is switched off, the Scenario Manager sends a stopTxToMobile event in the RAT in which a session is running and RemoveMobile events to all RATs in which the mobile is associated.

Detailed Design and Implementation of Each Functional Entity

Considering the architecture described in Section II, along with the events defined in Section III, the present Section outlines design and implementation details about each functional block of the proposed testbed. We start by the VDT editor, we focus on the VDT controller and the parts of the Central Controller, and we conclude with the description of the RAT VDT module.

VDT Editor

The VDT Editor is the end user interface through which the user creates and executes simulations in the distributed testbed. The VDT editor, a snapshot of which is depicted in Fig. 4, is a 4MB executable running on Windows, free for downloading from any user. The whole application is developed in C++ using the libraries of the wxWidgets [Smart 2005] in order to achieve cross-platform interoperability.



```
16:30:12: Application started...Waiting for user action
16:30:19: Login into Unite Controller...
16:30:21: Login successful
16:30:21: User is authenticated
16:30:21: Initializing information downloading from Unite Controller
16:30:21: Downloading available VDT Modules
16:30:22:
```

FIG. 4. THE VDT EDITOR

The user can easily configure a distributed simulation through a Graphical User Interface. The steps required are to login in the VDT Editor, select the necessary modules (Scenario Manager, VDTDB, at least one RAT VDT module and, optionally, JRRM and other RAT modules), configure the initial parameters of each simulator and create the scenario story. The last is achieved with the Scenario Editor, where the user fills a table similar to the one of Fig. 2b and specifies connection of each mobile with the available RATs by filling the records of a table similar to this of Fig. 2a.

When the user creates a simulation plan, he can save it as a file on its local drive and retrieve it lately if necessary, or he can upload it for simulation. In either case, the VDT editor creates an XML configuration file with all the necessary information of the simulation. In case of uploading the scenario for simulation, the VDT editor communicates with the VDT Controller and sends the XML configuration file. The IP address and port of the VDT controller is pre-configured in the VDT Editor, but can be changed by the user. After the simulation plan has been uploaded, the VDT Controller starts its compilation and execution of the simulation.

VDT Controller and Central Controller

The VDT Controller and the Central Controller run on the same hardware machine. They actually constitute two separate servers running on Linux. The first serves the VDT Editor, while the second serves the RAT VDT modules. Both, the VDT Controller and the Central Controller are implemented in Java with the use of the TEA framework [Tea].

The VDT Controller is continuously waiting for connections from the VDT editor. Once it receives a connection (actually a login request), it authenticates the user and provides information to the VDT Editor for the available RATs and RRM algorithms. After request of the VDT Editor, it receives the XML configuration scenario and parses it, transforming XML information to execution code. After that, it triggers the Scenario Manager to start the simulation. Note that objects created from the VDT Controller are fully accessible from the Scenario Manager for executing the simulation scenario.

The Central Controller on the other hand, is the functional entity for controlling the overall simulation. As mentioned in Section II, it is constituted of the Scenario Manager, the VDTDB and the JRRM, if the latter is implemented as a part of the Central Controller. Each one of the aforementioned entities is described below.

Scenario Manager

The Scenario Manager is in charge of logically integrating simulators of different radio technologies and presenting to the end-user a virtual distributed simulator of a composite network consisting of heterogeneous segments. It takes input from the VDT Controller in the form of simulation plans edited by the end-users. The information contained in the simulation plans corresponds to configuration parameters that are not changed during the simulation run-time (e.g., it is assumed that session arrival events and mobility patterns are independent of the state of the simulation). After loading the scenario description, the Scenario Manager starts the simulation in terms of a time-based loop.

When the Scenario Manager reads the scenario description provided by the VDT Controller, it starts the simulation by sending a Start event to all simulators. This event contains also all the configuration parameters for each simulator. Then it starts executing a loop for every time step defined, until the end of the simulation. Inside this loop, the following actions are performed: First it reads the scenario story and checks if a new mobile is to be added at the specific time instance. For any mobile to be added, it sends an AddMobile and a SendStreamToMobile event in every simulator the mobile is supposed to be connected to (obtained from the terminals configuration table), and updates the mobile IDs in the terminals mapping table of the VDTDB. It then asks the JRRM module through the appropriate event for the RAT where the session will be initiated (if possible). After the response of the JRRM it sends a StartTxToMobile to the appropriate RAT VDT module in order the session execution to start. Note that the second parameter of this event is now the local ID of the mobile and not the global ID. When adding of all mobiles is completed, it checks for every mobile if a running session terminates. For any session terminating, it resets its OFF counter and sends a StopTxToMobile event to the appropriate VDT module. Afterwards, the Scenario Manager checks for every mobile if an idle session is to be initiated. For every session to be initiated it resets its ON counter and sends a StartTxToMobile event to the appropriate RAT. After session checking is complete, it checks if the resource optimization algorithm is to be executed at the specific time instance. If yes, it sends a PostProcessHandover event to the JRRM. The JRRM replies with a list of Handover objects. Each handover object contains parameters about the mobile, the source RAT and the destination RAT the handover refers to. For every handover indicated by the JRRM, the Scenario Manager sends a StopTxToMobile event to the source RAT VDT module and then a SendStreamToMobile and a StartTxToMobile event to the destination RAT VDT module.

After session initiation/termination and RRM actions are completed, the Scenario Manager sends a ProcessTime event to all simulators in order to simulate the next time step.

Note that, in the routing and communication layer, all of the above methods defined in the pseudo code are transformed in SOAP messages, sent to the VDT modules. On the other hand, every value returned by a method is transformed to a SOAP reply message from the corresponding VDT module.

VDTDB

The VDTDB is actually the module that provides the interface for communicating with the database. It transforms SOAP based messages to database transactions and maps event parameters to fields in the tables of the database.

The actual database is implemented in PostgreSQL. It is constituted of several tables, most of which are dynamically created and deleted after the end of the simulation, based on user configuration parameters (e.g. selected RATs, number of mobiles e.t.c.).

The VDTDB is called by the JRRM module whenever an algorithm needs simulation results in order to take a decision, by the Scenario Manager in order to update mapping tables and by the RAT VDT modules in order to store simulation results. The VDTDB is also responsible for storing permanent parameters related to network layer interconnection of the distributed modules (e.g. IP and port of each RAT VDT module). While the structure of the VDTDB is complex and vital for the proper operation of the proposed testbed, the detailed presentation of its tables is outside the scope of this paper.

JRRM

As explained in Section II, the proposed testbed provides some RRM algorithms already implemented in the Central Controller. From this point of view, the JRRM module can be considered as part of the Central Controller. However, the proposed architecture provides the ability for a user to implement and integrate its own RRM algorithm as an external module. In this case, the JRRM is considered as a separate, geographically independent module, like a RAT VDT module. Specifically, what is different in the JRRM compared to a RAT module is the Federated Service (RRM algorighm in the first case and simulator in the second) and the events exchanged.

In both above cases, the operation of the JRRM is the same, and the messages exchanged with the Scenario Manager and the VDTDB are also the same. Note that, if the JRRM is a part of the Central Controller, the events refer to system methods, while if the JRRM is a separate entity, the events are transformed to SOAP messages.

RAT VDT Module

As discussed in Section II, each RAT VDT module is responsible for simulating a wireless access network. It is constituted of the simulator itself, called Federated Service, and the Federated Gateway. The purpose of the first is, obviously, to simulate the wireless network and produce results. Every simulator attached to the proposed testbed must provide public methods for freezing the simulation when necessary, simulating for dynamically specified time intervals, producing results for the specific intervals right after their simulation, dynamically adding or removing mobile nodes and dynamically changing the traffic generation parameters of each node. Moreover, it must implement the service mapping table outlined in Fig. 2c in order to run the services indicated by the Scenario Manager.

The purpose of the Federated Gateway is to proceed to the appropriate actions and bind the appropriate methods of the simulator whenever an event is received, and to send the appropriate events to the rest of the modules when necessary. The Federated Gateway is written in the language of the simulator and, actually, it constitutes an extension of the simulator itself. For every event supported by the simulator, a method exists in the Federated Gateway that is bound when the event is received. The mapping of the event to the method is performed by an API and will be discussed in the next subsection. The method implements all the necessary actions that must be taken and binds all the necessary methods of the simulator upon reception of a specific event.

Interconnection of Modules and Routing of Events

All the discussion performed so far referred to the application layer. However, clusters are geographically distributed and they communicate with SOAP messages through the Internet. Towards this direction, despite the fact that the proposed testbed is distributed in the application layer, a centralized approach was adopted in the routing and communication layer. More specifically, all geographically distributed VDT modules communicate only with the Central Controller, which is responsible for routing the events it receives to the appropriate modules. Hence, in the communication layer, all modules are not aware of the existence of other modules, except the Central Controller from which they receive and to which they send events. From a first point of view, this centralized approach seems to violate the whole distributed approach of the proposed testbed. Still, consider that this centralized communication approach makes the whole testbed transparent to each module, making the implementation of the clusters and the management of events easier.

For the communication of the VDT modules with the Central Controller through SOAP, an API was developed both for UNIX and Windows operating systems. In order to have cross-platform interoperability, the gSOAP framework [Engelen 2002] was used for implementing SOAP based communication on the API. Each simulator uses the libraries and the provided functions of the API in order to send/receive events, making thus the network layer communication transparent. The API provides an XML configuration file where the IP and the port of the Central Controller are defined. In the same file, the simulator developer must define, for each event received, the method of the Federated Gateway that will be bounded. Moreover, several methods are provided by the API for sending/receiving events, for adding parameters to events and for extracting parameters from events. Still, note that the correct matching of the parameter names and types between the source and destination modules is in the responsibility of the developers of these modules. On the other hand, considering that the geographical independence of the clusters requires communication through the Internet, communication error correction mechanisms are implemented as well in order the simulation not to be affected by temporal communication errors. Finally, note that the overall approach implemented in the API, is obviously embodied in the Central Controller as well.

Conclusions

The paper presents the components of a geographically distributed simulation platform that allows for testing of inter-system optimization algorithms in composite wireless networks. The overall architecture of the testbed is discussed and detailed design implementations are outlined together with the respective validation of the communication semantics that prove that the proposed testbed is suitable for time consuming simulations incorporating complex scenarios. A large number of validation results derived through a big number of tests could be found at [Sarakis 2009]. VDT answers well to the basic challenge for a future Internet experimental facility that provides a testing platform that is generic enough to allow experimentation in a technology environment with high diversity.

REFERENCES

Engelen R Van, K.Gallivan, “The gSOAP ToolKit for Web Services and Peer-To-Peer Computing Networks”, in Proc.CCGrid2002, Berlin, Germany, May 2002

Grau A. et.al, “Time Jails: A Hybrid Approach to Scalable

Network Emulation", in Procs. IEEE PADS '08, Rome, Italy, Jun. 2008

Hu W., H. Sarjoughian, "Discrete-Event Simulation of Network Systems Using Distributed Object Computing Hybrid Discrete", in Proc. SPECTS '05, Philadelphia, USA, 2005

IEEE Std. 1516-2000 –Standard for Modelling and Simulation (M&S) High Level Architecture (HLA)-Framework and Rules

Riley G. et. al., "A Federated Approach to Distributed Network Simulation" ACM Transactions on Modeling and Computer Simulation, vol.14, no.2, Apr. 2004, pp.116-148

Sarakis L., and G. Kormentzas, ''Performance Evaluation of UNITE Virtual Distributed Testbed'' In Proc. 6th International Workshop on Next Generation Networking Middleware (NGNM09), pages 91-103, Oct. 2009

Smart J., K.Hock, S.Csomor, "Cross Platform GUI Programming with wxWidgets", wxWidgets Developer Team, 2005

"The Tea Scripting Language", Information available online at http://www.pdmfc.com/tea-site/info

Georgios T. Kormentzas is currently Assistant Professor in the University of the Aegean, Department. of Information and Communication Systems Engineering. George Kormentzas was born in Athens, Greece in 1973. He received the Diploma in Electrical and Computer Engineering and the Ph.D. in Computer Engineering both from the National Technical University of Athens (NTUA), Greece, in 1995 and 2000, respectively. He has been actively working for many years on the area of network management and quality of service of computer and communication systems where he has introduced the concept of abstract information model, an ancestor of next generation networking middleware, which constitutes his main current research interest along with event-based distributed systems. He has published extensively in international scientific journals, edited books and conference proceedings. He is a member of pronounced professional societies, an active reviewer for several journals and conferences and EU-Independent Expert (being both Evaluator and Rapporteur) for various EC framework Programmes. Furthermore, George Kormentzas acts within Technical Program and Organizing Committees for numerous conferences and workshops and as a Guest Editor for scientific journals. At 2008, he served as General Chair of 4th IEEE/IFIP International Week on Management of Networks and Services and TPC co-chair of 19th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM). For three years (2010-2012), George Kormentzas acted as Vice President of OPEKEPE, the Greek Paying Agency for Community Aids according to the EC Common Agriculture Policy (CAP).